DIGITAL ACCESS TO SCHOLARSHIP AT HARVARD
DASH.HARVARD.EDU

HARVARD LIBRARY
Office for Scholarly Communication

A New Algorithm for View-Dependent Optimization of Terrain with Sub-Linear Time CPU Processing

Citation
Zhu, Yuanchen. 2008. A New Algorithm for View-Dependent Optimization of Terrain with Sub-Linear Time CPU Processing. Harvard Computer Science Group Technical Report TR-07-08.

Permanent link
http://nrs.harvard.edu/urn-3:HUL.InstRepos:24019789

Terms of Use
This article was downloaded from Harvard University's DASH repository, and is made available under the terms and conditions applicable to Other Posted Material, as set forth at http://nrs.harvard.edu/urn-3:HUL.InstRepos:dash.current.terms-of-use#LAA

Share Your Story

The Harvard community has made this article openly available.
Please share how this access benefits you. Submit a story .

Accessibility

A New Algorithm for View-Dependent Optimization of Terrain with Sub-Linear Time CPU Processing

Yuanchen Zhu

TR-07-08

Computer Science Group
Harvard University
Cambridge, Massachusetts

A New Algorithm for View-Dependent Optimization of Terrain with Sub-Linear Time CPU Processing

Yuanchen Zhu, *Student Member, IEEE Computer Society*

***Abstract*— This paper presents new schemes for view-dependent continuous level-of-detail (LOD) rendering of terrain which update output meshes with sub-linear CPU processing.**

We use a *directed acyclic graph* (DAG) abstraction for the longest-edge-bisection based multiresolution model. The other component of our refinement framework is the saturated monotonic perspective-division based error function. We made the critical observation that, for a vertex, the difference between the reciprocals of this particular error function for two different viewpoints is bounded by the distance between the two viewpoints, times a per-vertex constant. We call this the *bounded variation* property.

Utilizing this property, we introduce the *distance deferral table*, a circular array based structure that schedules deferred processing of DAG vertices according to viewpoint motion. We then use the distance deferral table to optimize the traditional threshold-based refinement algorithm and the dual-queue constrained optimization algorithm to allow sub-linear CPU run-time.

***Index Terms*— Viewing algorithms, virtual reality, visualization techniques and methodologies, terrain visualization, continuous level-of-detail, view-dependent optimization, deferred processing, multiresolution representation**

I. Introduction

REAL-TIME visualization of large-scale terrain models is at the core of display systems for many interactive applications including flight simulators, geographic information systems, and electronic games with vast outdoor environments. To accommodate the large geometry complexity of typical terrains while still maintaining high frame-rates, algorithms for view-dependent level-of-detail (LOD) triangulation of terrain are needed.

Typical terrain models have both relatively rough and flat neighborhoods. Moreover, distant areas affect the quality of screen image less than nearby areas due to perspective projection. View-dependent terrain LOD schemes take both into account and provide different tessellation levels for different parts of the terrain. This also means that the terrain has to be dynamically triangulated at run-time, since the distance from the viewpoint to a section of the terrain changes as the viewpoint moves. This paper presents a new method for performing view-dependent LOD triangulation of terrains which, most notably, is capable of updating the triangulation in sub-linear CPU run-time. While our work is currently only specified for the longest-edge-bisection based multiresolution representation, it should be straight forward to adapt it to view-dependent refinement of other multiresolution representations that are based on selecting a subset from a pool of LOD nodes.

At the core of the new method is the observation that for the commonly used view-dependent error metric, based on the quotient between a constant (often a view-independent measure of error) and the distance between the viewpoint and the point in question, there exists a constant on the fluctuation of the reciprocal of the metric over the distance traveled by the viewpoint. We do note, however, that our scheme is dependent on this property and, hence, restricts the class of error functions that can be used. In practice, however, a lot of the recent methods, such as [1]–[4], use this kind of error function.

The following are the specific contributions of our work.

- *Dual-table thresholding*. A simple and robust algorithm for thresholding based view-dependent-refinement which has sub-linear CPU run-time is presented. In comparison, existing schemes need to evaluate the error function for all potential valid refinement and is, hence, output sensitive.
- *Dual-queue constrained optimization*. An optimization to the classic binary heap allows the classic dual-queue refinement algorithm proposed in [5] to operate in $O(m + \Delta n \log n)$ time, where n is the output size, $m \ll n$ is a factor dependent on the current speed of the viewpoint, and $\Delta n \ll n$ is the difference between the output of successive frames. In comparison, the classic dual-queue scheme still needs to evaluate the error function for all entries in the priority queues and, hence, runs in linear time.
- *Distance deferral table*. We propose the *distance deferral table* which is capable of exploiting the bounded variation property of the error function. The scheme is very general and can be used to accelerate other refinement schemes. One example

is the *bucket queue* [6] based implementation of the dual-queue algorithm, which removes the $\log n$ factor from the running time. The minimal distance that the viewpoint needs to travel before the vertex switches buckets can be used as the index of the vertex within the distance deferral table.

In addition, our implementation leverages existing techniques such as *aggregated LOD handling* and *caching* on the graphics memory of [2] to improve performance.

II. Related Work

View-dependent level-of-detail algorithms usually decompose the input model into a multiresolution representation and then, at run time, extract the appropriate details to include in the view-dependent reconstruction of the original model. In this section, we summarize some of the recent works and focus on different multiresolution representations and algorithms for choosing the appropriate details.

A. Multiresolution Representation

Triangular irregular networks. Much of the earlier works on terrain LOD algorithms have concentrated on discrete multiresolution *triangulated irregular networks* (TIN) modeling. Several versions of the same landscape tessellated to different detail levels are produced with expensive off-line preprocessing and stored in some way. During run-time, they are dynamically combined to provide different tessellation levels for different areas of the terrain. Examples are [7]–[9]. For these schemes, refinement and simplification are relatively coarse grained, but the required CPU work is also minimized, as each LOD operation affects a cluster of triangles instead of an individual triangle.

Longest-edge-bisection. For terrain, Duchaineau et al. [5] have proposed using the *binary triangle tree*, based on the *longest-edge-bisection* operation, as the base of refinement framework. A lot of recent work, such as [1]–[3], [10], also uses this representation. The resultant triangulation is called *right-triangular irregular network* by Evans et al. [11]. Initially a square domain is covered by two right triangle that joins at the hypotenuse. A Longest-edge-bisection repeatedly bisects the shared hypotenuse of a pair of triangles, and a right-triangular mesh of variable resolution can be created this way. Due to the regular nature of such a hierarchy, this multiresolution representation is particularly easy and efficient to implement.

Progressive mesh. Hoppe [12] has presented the *progressive mesh* (PM) structure that represents an arbitrary input mesh as a coarse base mesh and a sequence of *vertex splits* operations which are inverse of *edge collapses*. Earlier work by El-Sana and Varshney [13] and later work by Kim and Lee [14] use the same edge collapse operation, but have different condition for interdependency among the vertex splits. View-dependent selective refinement can also be performed [13]–[15]. Hoppe has also adapted his system to terrain rendering [16], although the flexibility offered by irregular mesh refinement also makes implementing PM slightly more cumbersome and less efficient than longest-edge-bisection based schemes.

Hybrid representation. A common problem with implementing longest-edge-bisection and progressive mesh based view-dependent algorithms is that these schemes have become increasingly CPU bound and, hence, inefficient in the face of modern graphics hardware. Several hybrid schemes that combine two types of multiresolution representation have been recently proposed to combat this problem. Pomeranz [10] and Levenberg [2] have both suggested staying within the longest-edge-bisection framework, but using triangle patches in place of individual triangles in the binary triangle tree of [5]. Cignoni et al. [3] go one step further and replace individual triangles with precomputed TINs that agree on bisection boundary. Yoon et al. [17] use a *cluster hierarchy* similar to to Erikson et al.'s *hierarchical LOD* [18] to represent the original model, but store each node in the hierarchy as an individually refined PM. Finally, Zhu [19] employs the view-dependent PM refinement framework, but replaces each triangle in the PM reconstruction with a uniform triangle patch generated through remeshing using a precomputed parametrization. A common attribute of all of these schemes is that they use a relatively elaborate multiresolution representation at the macro-level and a much simpler representation at the micro-level. This allows compromise to be made between the fine-grainedness of the LOD adjustments and the required CPU processing time. Also the corresponding micro level representations usually allow more efficient hardware rendering due to improved batching and graphics hardware cache utilization. Our proposed schemes are well suited for handling the micro level refinement.

B. Refinement Algorithm

Top-down refinement. Many previous works including [1], [3], [20], [21] use a top-down refinement scheme. In particular, Lindstrom and Pascucci [1] employ a simple and stateless one-pass top-down traverse to triangulate the terrain. The benefit of such top-down approaches

is the ease of implementation. However, in the case of [1], [20], [21], the entire triangulation is regenerated for each frame, which requires CPU processing time linear in the size of the rendered mesh. Even in [3] where the final rendered mesh is cached between frames and only modified mesh parts are uploaded to the graphics memory, the top-down traverse still has a linear running time due to the need to calculate the error function for each node in the refinement hierarchy. However, *aggregated LOD handling*, as used in [2], [3], [10], can reduce the constant factor in the linear term by an arbitrary factor, although this comes at a price of coarser LOD adjustment.

Frame-coherent refinement. For arbitrary meshes, Hoppe [15] maintains an *active vertex front* and moves part of the front up or down the PM vertex hierarchy, which corresponds to performing *edge collapses* and *vertex splits* on the mesh. For terrain, Duchaineau et al. [5] use two priority queues to drive incremental refinement and simplification in their *real-time optimally adapting meshes* (ROAM) algorithm. Both Hoppe's and Duchaineau et al.'s schemes only perform the required refinement and simplification operations per frame. However, to ensure correctness, in both cases the view-dependent error function needs to be evaluated for each node in the vertex front or the priority queues. This again causes the total running time to be linear in the size of the output mesh.

Sub-linear time CPU processing. *Amortization*, initially proposed by Hoppe [15], is the most basic technique to further reduce the linear running time of view-dependent refinement algorithms. Instead of evaluating the error function for all nodes, only a small portion of them are processed for each frame. The running time is thus reduced accordingly. However, since many potential refinement and simplification operations are ignored for each frame, the resulting output mesh is also sub-optimal for a given frame. More recent work by El-sana and Bachmat [22] prioritizes this amortization process so that nodes with higher *energy* are traversed more often, where the energy of a node is proportional to its closeness to the viewpoint. Compared with our scheme, the energy based scheme is more general but only provides a rough hint, so mesh sub-optimality can still occur. The authors of the ROAM algorithm [5] also propose using a bound of the screen-distortion priority over time. In comparison, our scheme uses a bound over viewpoint motion. ROAM's approach is somewhat indirect in that they suggest using an upper bound of viewpoint speed and using it to calculate the priority fluctuation bound over time. On the other hand, we are able to directly derive a priority fluctuation bound over viewpoint speed. This results in improved robustness and efficiency since the viewpoint is not always moving at the highest speed. To our knowledge, currently no publicly available implementation of the ROAM algorithm actually uses the deferred priority computation scheme proposed in [5]. Finally, Zhu [23] has presented a similar circular array based scheme as ours, but his scheme requires the screen-space error threshold to be fixed during processing. Although a method for dynamically adjusting the output mesh complexity is provided, it is not really correct in the sense that the adjustment causes different screen-space error thresholds to be used for different nodes.

III. Overview

In the rest of this paper, we explain the various parts of our algorithm: the basic refinement framework, the dual-table refinement algorithm, and the dual-priority refinement algorithm.

Our refinement framework is based on the commonly used *longest-edge-bisection* operation (Section IV-A). For simplicity and easier adaption to other multiresolution representation, we use its *directed acyclic graph* abstraction (Section IV-B) to describe our schemes. The other component of the refinement framework is the error function (Section IV-C), which serves as the guide for adaptive refinement. In our work, we use Lindstrom and Pascucci's isotropic error function [1] since it is the straight-forward saturation of the basic perspective-division error function. This error function satisfies a special property that allows a significant optimization to be made, which we will discuss in Section V-B. In Section V-A and Section VI-A, we review the basic algorithm for performing *thresholding* and *constrained optimization* based adaptive refinement, which, unfortunately, have output sensitive running time. Then in Section V-C and Section VI-B, we present the optimization that allows these two schemes to achieve sub-linear CPU run-time, utilizing what we call the *distance deferral table*. Finally in Section VII, we present empirical results and discussions of our current implementation of the presented algorithms.

IV. Refinement Framework

A. Longest-Edge-Bisection

At the base of our refinement algorithm is the commonly used *longest-edge-bisection* operator (see Fig. 1), which can be used to partition a unit square into right isosceles triangles of various resolution. Initially, the domain is covered by a pair of right triangles separated by the main diagonal of the square. Each successive longest-edge-bisection then picks a pair of right triangles

that share one edge as their hypotenuses (or a single triangle if its hypotenuse is on the boundary), and bisects this edge to create four (or two in the boundary case) smaller right triangles. Such paired triangles are called *diamonds*, and each diamond can be uniquely identified by the new vertex formed by the bisection. The inverse of longest-edge-bisection is simply a *diamond merge*, which is also illustrated in Fig. 1

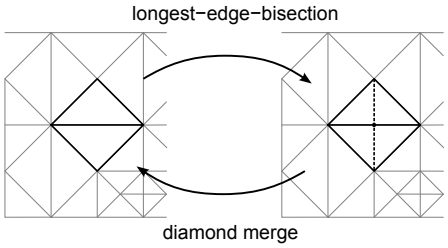

Fig. 1. Longest-edge-bisection and its inverse operation.

As each bisection takes place, a diamond formed by two triangles is replaced by four smaller triangles, which themselves are parts of smaller diamonds. This induces a dependency relationship between diamonds. This dependency is illustrated in Fig. 2, which represents diamonds by the corresponding vertices. A diamond cannot be bisected until the diamonds it depends on have been bisected, in which case we say that the bisection associated with the diamond is a *legal refinement*. Similarly, a bisected diamond cannot be merged until all the smaller diamonds depending on it have been merged, in which case we say that the corresponding merge is a *legal simplification*. If a set of diamonds satisfy that, for each diamond in it, all of the diamonds it depend on also belong to the set, then we say that the set is *legal*.

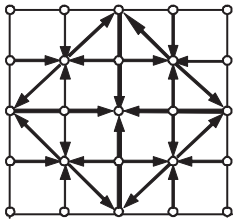

Fig. 2. Dependency between diamonds. Diamonds are represented by the corresponding vertices.

Intuitively we can overlay the initial unit square onto a $2^n + 1$ by $2^n + 1$ rectangular vertex grid. Each diamond is then uniquely associated with a vertex entry in the grid. If we start with one diamond covering the entire grid and repeatedly perform longest-edge-bisections until all diamonds that have corresponding grid vertices have been bisected, then the mesh we arrive at represents the vertex grid triangulated at full resolution. Let M be the set of all these diamonds. Then if $S \subset M$ is legal, then it corresponds to a simplification of the initial grid, and we say that S is a *legal* reconstruction. Clearly, the set M together with the diamond dependency thus gives us a multiresolution representation of the initial vertex grid.

B. DAG Abstraction

The longest-edge-bisection based multiresolution representation can be conveniently abstracted by a directed acyclic graph, $G = (V, E)$, where V is the vertex set and E is the edge set. The vertices of the DAG correspond to the diamonds and the edges correspond to the dependencies between the diamonds. To simplify notation, for a subset $S \subset V$, we define $\lfloor S \rfloor = \{u \in V : \exists v \in S \text{ s.t. } (u, v) \in E\}$ and $\lceil S \rceil = \{u \in V : \exists v \in S \text{ s.t. } (v, u) \in E\}$. Also, if $v \in V$, then we write $\lfloor v \rfloor = \lceil \{v\} \rceil$ and $\lfloor v \rfloor = \lfloor \{v\} \rfloor$.

Using these notations, a legal reconstruction of the longest-edge-bisection multiresolution representation is a DAG vertex subset $R \subset V$ satisfying $\lfloor R \rfloor \subset R$. A legal simplification operation then removes one vertex from R while maintaining the legality of R. It follows that only a vertex in $R - \lfloor R \rfloor$ can be legally removed, because all other vertices in R have descendant vertices that are still in R. We denote this subset of R as R^-, and call it the *simplification candidates*. Likewise, for a legal refinement operation, only vertices in $R^C - \lceil R^C \rceil$ (where $R^C = V - R$) can be removed. We denote this subset as R^+, and name it the *refinement candidates*.

We will present the rest of our work using the above abstraction and revert to triangles and diamonds only when necessary.

C. Error Function

In typical top-down refinement, the application starts with the coarsest reconstruction, containing only the DAG vertex corresponding to the top most diamond, and repeatedly choose one vertex from the current refinement candidates to add to the current reconstruction, until some kind of accuracy or complexity threshold is met.

A view-dependent refinement algorithm needs to decide exactly which vertex to refine from the set of refinement candidates for each refinement step, as well as which vertex to simplify from the set of simplification candidates for each simplification step. These decisions are usually guided by a view-dependent error function, $\epsilon(v, \mathbf{e})$, where v is the vertex in question and $\mathbf{e}$ is the viewpoint. For a given viewpoint, the error function assigns to each DAG vertex a number that estimates the relative viewing error introduced if the reconstruction doesn't include the corresponding vertex. Vertices with

greater errors should be included in the final reconstruction before those with smaller errors. Also, for any vertex in a legal reconstruction, all its ancestor vertices should also be in the reconstruction. Hence it is natural to require the error function to be *monotonic*, i.e., for an arbitrary viewpoint $\mathbf{e}$, the error function satisfies $\epsilon(u,\mathbf{e}) \geq \epsilon(v,\mathbf{e})$ for all $(u,v) \in E$.

The most commonly used view-dependent error metric is based on perspective projection. Each vertex v is associated with a reference point $\mathbf{p}_v \in \mathbf{R}^3$ and a view-independent error measurement ξ_v. The view-dependent error is then approximated by $\xi_v/\|\mathbf{p}_v - \mathbf{e}\|$. In general, however, this error function does not satisfy the monotonicity requirement. Pajarola [21] has proposed saturating the view-independent error while Lindstrom and Pascucci [1] uses a bounding sphere hierarchy to saturate the view-dependent error: to each vertex v, a bounding sphere of radius r_v is assigned such that $B(\mathbf{p}_v, r_v) \subset B(\mathbf{p}_u, r_u)$ for any $(u,v) \in E$, where $B(\mathbf{p}_v, r_v)$ denotes the ball with radius r_v centered at $\mathbf{p}_v$. The view-independent error is also propagated such that $\xi_u \geq \xi_v$ for any edge (u,v). The error function which satisfies the monotonicity requirement is now defined as:

$$\epsilon(v,\mathbf{e}) = \begin{cases} \dfrac{\xi_v}{\|\mathbf{p}_v - \mathbf{e}\| - r_v} & \mathbf{e} \notin B(\mathbf{p}_v, r_v) \\ +\infty & \mathbf{e} \in B(\mathbf{p}_v, r_v) \end{cases} \quad (1)$$

For reasons that will become clear later, in our application, instead of using the above error function as is, we use its reciprocal. We write the new function as $\kappa(v,\mathbf{e})$ and call it the *priority function*. In practice, this means that vertices with *smaller* priority values should be included in the reconstruction. The new function is now

$$\kappa(v,\mathbf{e}) = \begin{cases} \dfrac{1}{\xi_v}(\|\mathbf{p}_v - \mathbf{e}\| - r_v) & \mathbf{e} \notin B(\mathbf{p}_v, r_v) \\ 0 & \mathbf{e} \in B(\mathbf{p}_v, r_v) \end{cases} \quad (2)$$

V. Threshold Based Adaptive Refinement

A. Dual-Set Refinement

The most basic form of adaptive refinement is based on *thresholding*. The application defines a threshold priority value μ that governs the minimally allowed priority values. All vertices with higher priority values are discarded from the reconstruction, so the desired view-dependent reconstruction is specified by $\{v \in V : \kappa(v,\mathbf{e}) \leq \mu\}$. Note that it is more customary to specify a screen-space threshold error τ and let $\mu = 1/\tau$. For a vertex v and a viewpoint $\mathbf{e}$, we say that v is *enabled* if v is included in the reconstruction, and disabled if otherwise.

Utilizing temporal coherence, a basic implementation can maintain the current reconstruction R, the simplification candidates R^-, and the refinement candidates R^+. For each frame, we perform refinements and simplifications on R using vertices from R^- and R^+.

Unfortunately, the algorithm still needs to evaluate the priority function for at least every vertex in R^- and R^+ in order to decide which vertices to add or remove. Due to the regular hierarchical nature of the DAG, both $|R^-|$ and $|R^+|$ are on the order of $O(|R|)$. Hence the algorithm still requires output-sensitive running time, although the number of actual refinement or simplification operations performed indeed only depends on the difference between the output of consecutive frames.

B. Bounded Variation of Priority Function

We now make the crucial observation that for a vertex v and two different viewpoints $\mathbf{e}, \mathbf{e}' \notin B(\mathbf{p}_v, r_v)$,

$$\begin{aligned} |\kappa(v,\mathbf{e}) - \kappa(v,\mathbf{e}')| &= \frac{1}{\xi_v}\left|\|\mathbf{p}_v - \mathbf{e}\| - \|\mathbf{p}_v - \mathbf{e}'\|\right| \\ &\leq \|\mathbf{e} - \mathbf{e}'\|/\xi_v, \end{aligned} \quad (3)$$

Also, even if any of the two viewpoints is in $B(\mathbf{p}_v, r_v)$, (3) is still satisfied. The in-equation thus implies that, as the viewpoint travels through space, the fluctuation of the priority of a vertex is bounded by the distance traveled by the viewpoint divided by the constant ξ_v. We say that such a priority function is of *bounded variation*.

Suppose the vertex v is enabled for viewpoint $\mathbf{e}'$. Then a necessary condition on the viewpoint $\mathbf{e}$ for which v is disabled is

$$\begin{aligned} \kappa(v,\mathbf{e}) > \mu &\Rightarrow \kappa(v,\mathbf{e}) - \kappa(v,\mathbf{e}') > \mu - \kappa(v,\mathbf{e}') \\ &\Rightarrow \left|\kappa(v,\mathbf{e}) - \kappa(v,\mathbf{e}')\right| > \mu - \kappa(v,\mathbf{e}') \\ &\Rightarrow \|\mathbf{e} - \mathbf{e}'\| > \xi_v(\mu - \kappa(v,\mathbf{e}')). \end{aligned} \quad (4)$$

Similarly, suppose v is disabled for viewpoint $\mathbf{e}'$. If it is to be enabled, the new viewpoint $\mathbf{e}$ satisfies

$$\|\mathbf{e} - \mathbf{e}'\| \geq \xi_v(\kappa(v,\mathbf{e}') - \mu). \quad (5)$$

Combining (4) and (5), we observe that a vertex can toggle its status only when the distance between the new viewpoint $\mathbf{e}$ and the old viewpoint $\mathbf{e}'$ satisfies $\|\mathbf{e}-\mathbf{e}'\| > \xi_v\,|\kappa(v,\mathbf{e}) - \mu|$. Moreover, suppose the viewpoint is at $\mathbf{e}_0$ in frame 0 and moves to $\mathbf{e}_1, \mathbf{e}_2, \mathbf{e}_3, \ldots$ in the following frames. If the vertex toggles its status in frame n, then

$$\begin{aligned} \xi_v\,|\kappa(v,\mathbf{e}_0) - \mu| &\leq \|\mathbf{e}_0 - \mathbf{e}_n\| \\ &\leq \|\mathbf{e}_0 - \mathbf{e}_1\| + \ldots + \|\mathbf{e}_{n+1} - \mathbf{e}_n\|. \end{aligned} \quad (6)$$

In other words, whenever we calculate $\kappa(v,\mathbf{e})$ and either include or exclude v in R, we know that the status

of the vertex will not change until the total length of the poly-line formed by the viewpoint in the following frames becomes greater than $\xi_v \left|\kappa(v, \mathbf{e}) - \mu\right|$. Thus we can safely defer the processing of v until then.

C. Implementation Using Distance Deferral Table

The bounded variation property of the priority function admits a particular simple and robust thresholding algorithm with sub-linear CPU run-time. We exploit this property through a circular array, which we call the *distance deferral table*. Suppose the array is T and has $|T|$ entries. We associate with it a head pointer $\textsc{Head}(T) \in \{0 \ldots |T| - 1\}$ and a maximal range parameter d_T. The entries in the array are buckets of vertices. The algorithm maintains R^- and R^+ as two distance deferral tables. For each frame, suppose that the viewpoint from the last frame is $\mathbf{e}'$ and the current viewpoint is $\mathbf{e}$. The head pointer is then increased by $1 + \lfloor(|T| - 1)\,\mathrm{clamp}(\|\mathbf{e}' - \mathbf{e}\|/d_T, 0, 1)\rfloor$ and possibly wrapped around. Only vertices belonging to the buckets covered by the movement of the head pointer are taken out and processed. To insert a vertex v into the table, we linearly discretize the minimal distance that the viewpoint needs to travel into the integral index $\lfloor(|T| - 1)\,\mathrm{clamp}(\xi_v \left|\kappa(v, \mathbf{e}) - \mu\right| / d_T, 0, 1)\rfloor$. This index (offseted by $\textsc{Head}(T)$) gives the final bucket that v will be added into. The complete pseudo-code is listed in Table I.

The distance deferral table scheme can also be modified to allow "lagged-behind" processing. When the user is moving very fast, the amount of evaluation needed increases. However, to satisfy strict frame-rates, we can stop processing when the allotted time is to expire. This is achieved by only advancing $\textsc{Head}(T)$ by one entry at a time and performing the required processing. This is repeated for $1 + \lfloor(|T| - 1)\,\mathrm{clamp}(\|\mathbf{e}' - \mathbf{e}\|/d_T, 0, 1)\rfloor$ steps, or until the alloted time expires. Note that the same philosophy lies behind ROAM's *progressive optimization* [5].

D. Culling

To further improve the rendering performance, it is often customary to incorporate a "cull" factor into the priority function. If the block of geometry corresponding to a DAG vertex is outside of the view frustum or occluded by some other geometry (e.g., using the algorithm in [24]), then the priority value of that vertex should be set to ∞. This, of course, will break the bounded variation property. Hence, at the start of the frame, our algorithm performs a hierarchical traverse of the current reconstruction, similar to the one performed in [5] and [1]: If a previous unculled vertex is culled, then all vertices depending on it are removed from the reconstruction. If a previously culled vertex is unculled, then we add it to the set S_o in Table I. If a previously completely unculled vertex is still completely unculled, we stop the recursive traverse for all vertices depending on it. Otherwise, we continue the recursive traversal.

TABLE I
OPTIMIZED THRESHOLDING ALGORITHM

```
UPDATE-TABLE(T, l)
 1  n ← 1 + ⌊(|T| − 1) clamp(l/d_T, 0, 1)⌋
 2  S ← ⋃_{i=0}^{n−1} T[HEAD(T) + i mod |T|]
 3  Clear T[HEAD(T) + i mod |T|] for i = 0 ... n − 1
 4  HEAD(T) ← HEAD(T) + n mod |T|
 5  return S

PUSH-TABLE(T, v, l)
 1  i ← HEAD(T) + ⌊(|T| − 1) clamp(l/d_T, 0, 1)⌋ mod |T|
 2  T[i] ← T[i] ∪ {v}

THRESHOLD-OPTIMIZE(R, T_I, T_O, e', e)
 1  S_I ← UPDATE-TABLE(T_I, ‖e' − e‖)
 2  S_O ← UPDATE-TABLE(T_O, ‖e' − e‖)
 3  while S_I ≠ ∅ do
 4    v ← POP-SET(S_I)
 5    if κ(v, e) > μ then
 6      SIMPLIFY(R, v)
 7      Remove vertices in ⌈v⌉ from T_I
 8      S_I ← S_I ∪ {u ∈ ⌊v⌋ : ⌈u⌉ ⊂ V − R}
 9      PUSH-TABLE(T_O, v, ξ_v |κ(v, e) − μ|)
10    else
11      PUSH-TABLE(T_I, v, ξ_v |κ(v, e) − μ|)
12  while S_O ≠ ∅ do
13    v ← POP-SET(S_O)
14    if κ(v, e) ≤ μ then
15      REFINE(R, v)
16      Remove vertices in ⌊v⌋ from T_O
17      S_O ← S_O ∪ {u ∈ ⌈v⌉ : ⌊u⌋ ⊂ R}
18      PUSH-TABLE(T_I, v, ξ_v |κ(v, e) − μ|)
19    else
20      PUSH-TABLE(T_O, v, ξ_v |κ(v, e) − μ|)
```

VI. Constrained Optimal Refinement

A. Dual-Queue Refinement

Sometimes the application may want to place a constraint on the maximal complexity of the view-dependent reconstruction. A complexity function $\sigma(v) \geq 0$ is defined such that, given the vertex v, $\sigma(v)$ describes its complexity. For simplicity we also write $\sigma(S) = \sum_{v \in S} \sigma(v)$ for a subset $S \in V$. Given a target complexity constraint λ, the desired simplification is then one of the subsets R satisfying $\epsilon(u, \mathbf{e}) \geq \epsilon(v, \mathbf{e})$ for any $v \subset R$, $u \in R^C$ and $\sigma(R) + \sigma(u) > \lambda$ where u is the vertex in S^+ with the smalled priority value, i.e., we stop refinement when the reconstruction is about to

exceed the complexity constraint. In practice, $\sigma(v)$ can be set to the number of triangles that will be added to the reconstruction if v is refined, and λ to the target triangle budget.

Duchaineau et al. [5] have proposed a general optimization framework based on dual priority-queues. Strictly speaking, only vertices in R^- and R^+ should be maintained, so only these two sets need to be kept in the queue. However, the algorithm still faces the same problem as the thresholding based adaptive refinement algorithm. In order to determine the refinement candidate with the greatest priority and the simplification candidate with the smallest priority, the priority function needs to be evaluated for all vertices in R^- and R^+. In this section we present an optimization to the basic binary-heap based priority queue implementation that achieves sub-linear CPU running time.

B. Distance Deferral Heap

Similar to the case of thresholding based adaptive refinement, we want to defer as much of the processing as possible. Recall that a binary heap is a complete binary tree satisfying that the priority of a parent node is always equal to or smaller (or greater for a maximal heap) than that of its child nodes. The bulk of the processing involved is due to evaluation of the priority function for all vertices in R^- and R^+ and the possible sifting (to ensure heap property) afterward. As long as the heap property is maintained, the nodes with smallest priority value can be pulled out in logarithmic time.

A necessary condition for violating the heap property is that, for an edge of the binary heap, (1) either of the two ends changes its position in the heap, or (2) the total priority fluctuation of the vertices at the two ends surpasses their difference in priority.

To check for the first condition, we perform heap operations such as HEAPIFY, ENQUEUE, and DEQUEUE as usual, but also maintain a vertex set M which is initially set to $\emptyset$ at the start of the frame. Whenever a heap operation changes the position of a vertex within the binary hierarchy, we add it to M.

To check for the second condition, we keep a reference of all non-leaf vertices inside a distance deferral table as presented in Section V-C. To calculate the relative index of the vertex within the table, we note that for two vertices u, v, and viewpoint $\mathbf{e}'$ s.t. $\kappa(u,\mathbf{e}') < \kappa(v,\mathbf{e}')$, in order to reverse their priority, the new viewpoint $\mathbf{e}$ must satisfy

$$\begin{aligned}
&\kappa(u,\mathbf{e}) - \kappa(v,\mathbf{e}) \geq 0 \\
\Rightarrow\ &(\kappa(u,\mathbf{e}) - \kappa(u,\mathbf{e}')) + (\kappa(v,\mathbf{e}') - \kappa(v,\mathbf{e})) \geq \\
&\quad \kappa(v,\mathbf{e}') - \kappa(u,\mathbf{e}') \\
\Rightarrow\ &\|\mathbf{e}-\mathbf{e}'\|/\xi_u + \|\mathbf{e}-\mathbf{e}'\|/\xi_v \geq \kappa(v,\mathbf{e}') - \kappa(u,\mathbf{e}') \\
\Rightarrow\ &\|\mathbf{e}-\mathbf{e}'\| \geq \frac{\xi_u\xi_v}{\xi_u+\xi_v}(\kappa(v,\mathbf{e}') - \kappa(u,\mathbf{e}'))
\end{aligned} \tag{7}$$

We write SAFE-DIST$(u,v,\mathbf{e}) = \xi_u\xi_v(\kappa(v,\mathbf{e}') - \kappa(u,\mathbf{e}'))/(\xi_u+\xi_v)$. At the start of each frame, we use UPDATE-TABLE on our distance deferral table, T, to get the list of vertices that require sifting. Then we perform usual heap operations and also maintain M. At the end of frame, we move all parent vertices in M into the T using the smaller of the vertex's SAFE-DIST with respect to its two child vertices as the argument to PUSH-TABLE. The pseudo-code is listed in Table II.

TABLE II
DISTANCE DEFERRAL HEAP ALGORITHM.

```
MARK-PATH(Q, i_0, i_1, T, M)
 1  for each v in the path between Q[i_0] and Q[i_1] do
 2    T ← T − {v}, M ← M ∪ {v}

ENQUEUE(Q, v, T, M)
 1  Append v to Q and sift v up as required
 2  MARK-PATH(Q, HEAP-INDEX(Q, v), |Q| − 1, T, M)

DEQUEUE(Q, v, T, M)
 1  u ← POP-BACK(Q)
 2  Remove v from T or M
 3  if u ≠ v then
 4    i ← HEAP-INDEX(Q, v)
 5    Set Q[i] ← u and sift u up/down as required
 6    MARK-PATH(Q, HEAP-INDEX(Q, u), i, T, M)

HEAPIFY(Q, T, l, M)
 1  S ← UPDATE-TABLE(T, l)
 2  while S ≠ ∅ do
 3    Pick a v ∈ S with the maximal tree depth
 4    i ← HEAP-INDEX(Q, v)
 5    Sift v down in Q as required
 6    MARK-PATH(Q, i, HEAP-INDEX(Q, v), T, M)
 7    S ← S − {v}
 8    if i ≠ HEAP-INDEX(Q, v) then
 9      S ← S ∪ {HEAP-PARENT(v)}
```

Notice that the arguments to MARK-PATH, i_0 and i_1, must have an ancestor-descendant relationship. Hence the queue index of vertices along the tree path from i_1 to i_0 can be calculated simply by shifting. This requires $O(\log n)$ running time, where n is the size of the heap. Hence both ENQUEUE and DEQUEUE have $O(\log n)$ running time. HEAPIFY will only sift a small subset of

parent vertices in the heap depending on the movement of the viewpoint, utilizing the distance deferral table.

Finally we note that a hierarchical traverse similar to the one presented in Section V-D can be used to support view-frustum culling and occlusion culling.

VII. Results and Discussions

Tests of our prototype implementation were done on a Pentium M 1.8GHz Dell D600 laptop with 1GB of RAM and ATI Radeon Mobility 9000 graphics card with 32M of VRAM installed on a AGP 4× port. For all results, a 2049×2049 16bit height field of the Grand Canyon was used. The flight course has a length of 4000 frames and contains sharp turnings and speed changes. We use a 800x600 viewpoint with 16bit color-depth and 24bit z-depth. The screen error threshold is set to 1 pixels. A 2048×2048 color map, chopped into textures of size 256×256, is applied along with a tiled high-frequency "detail texture", using simple multitexturing.

Our current implementation performs vertex-morphing on the CPU side using SSE instruction set, which means that the entire geometry has to be streamed into the graphics memory for each frame. The benefit is that we can better support earlier generation graphics hardware, since enabling vertex morphing can often allow us to tune up the screen error threshold to 4 pixels with no noticeable loss of visual quality. This in turn reduces the output size from over 150,000 triangles down to 60,000 triangles. On the other hand, our implementation is capable of rendering an output mesh containing over 230,000 triangles at over 80Hz on our testing platform, which corresponds to a throughput of 20 million triangles per second. In comparison, the 3DMark 2001 SE benchmark software by Futuremark Co. [25] reports a 16 million triangles per second throughput on the same hardware, so we don't yet find the dynamic streaming process to be a performance bottle-neck. However, for current and next generation graphics hardware, performing morphing using a vertex shader or not performing morphing at all is the preferred way to go. Our implementation also uses one step of "subdivision" for LOD handling, i.e., each triangle in the binary triangle tree corresponds to four sub-triangles in the output mesh.

Next we evaluate the efficiency of our scheme by plotting the frame time, the culling time, and the updating time for each frame, with and without the distance deferral table (DDT) optimization, as shown in Fig. 3. The updating time includes performing LOD refinement and simplification, but does not include vertex morphing since morphing is intermingled with rendering. For reference, the size of the output mesh is also plotted.

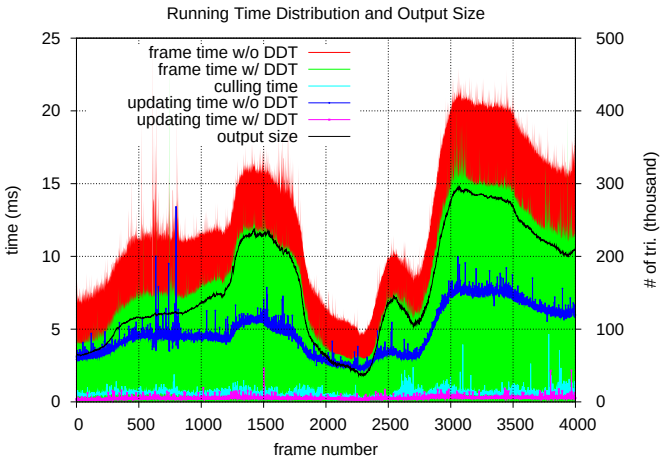

Fig. 3. Plots of frame time, culling time, updating time, and output mesh size, with and without the distance deferral optimization. The updating time includes LOD refinement and simplification, but does not include view-frustum culling and vertex morphing.

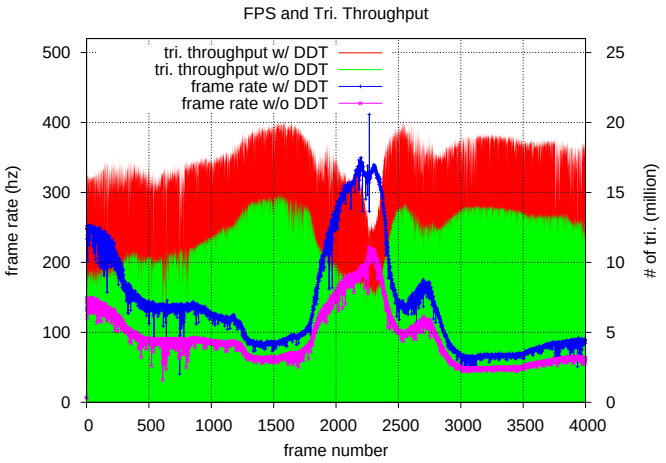

Fig. 4. Plots of frame rates and triangle thorough-put, with and without the distance deferral table optimization.

From the graph we can observe that in typical frames, the updating time using DDT optimization is usually around 3% of the updating time without the optimization. Also, the updating time using DDT optimization remains at a constant low level while, without the optimization, the updating time varies more or less linearly with the size of the output. The actual saving in over-all frame time is not as dramatic, at around 25% to 50%, largely because CPU processing and graphics hardware processing can happen in parallel. However, typical virtual reality applications will not want to use all CPU processing time for LOD control and can benefit from the extra three hundred or so milliseconds of CPU time saved for each second.

To demonstrate the actual rendering performance, we have also plotted the frame rates and the triangle

thorough-puts, with and without the DDT optimization in Fig. 4. The triangle thorough-put is simply calculated as frame-rate times the output size. Here we can notice that the through-put reaches relatively low levels when the output size is small and the frame rate is high. This is understandable since in such cases, the main limiting factor is the fill-rate of graphics hardware. When the frame rate is below 100Hz, the implementation can achieve a thorough-put of over 18 million triangles per second constantly. Compared with the unoptimized version, the distance deferral table achieves 30% to 70% increase in triangle thorough-put.

Finally we note that, in the case of constrained optimization, a similar boost in performance is observed when the distance deferral table optimization is applied. CPU side updating time typically decreases by 80% to 90%.

VIII. Conclusion and Future Work

We have presented our algorithms for performing view-dependent level-of-detail rendering of terrain with sub-linear CPU processing. Our algorithms work on the directed acyclic graph based abstraction of a multiresolution representation and can potentially be extended to other multiresolution representations. Our key observation is that the commonly used perspective-division based error function satisfies that the variation of its reciprocal is bounded by the distance between the viewpoints times a constant. In particular, our distance deferral table based thresholding algorithm exploits this property and is both robust and simple to implement. In practice, our algorithms significantly reduce the number of vertices that need to be processed for each frame while maintaining mesh optimality. Future works will include adapting the same schemes to the more irregular refinement hierarchy of progressive meshes and possibly supporting a wider variety of error functions.

Acknowledgements

This work is the more developed form of a project initially performed as an entry to the Intel International Science and Engineering Fair 2001. This manuscript was submitted for peer-review in 2005. I am grateful to my anonymous reviewers for their insightful comments and suggestions. However, during the long lapse between when I worked on the project and when I sought publication, another paper [26] that builds on some similar idea had been published. Also, as pointed out by one of the reviewers, with the improvement of graphics hardware, the problem addressed in the paper no longer represents a bottleneck in visualization. Hence I decide to make this manuscript public as is.

References

[1] P. Lindstrom and V. Pascucci, "Visualization of larget terrain made easy," in *IEEE Visualization 2001*, Oct. 2001, pp. 363–370.

[2] J. Levenberg, "Fast view-dependent level-of-detail rendering using cached geometry," in *IEEE Visualization 2002*, Oct. 2002.

[3] P. Cignoni, F. Ganovelli, E. G. F. Marton, F. Ponchio, and R. Scopigno, "BDAM: Batched dynamic adaptive meshes for high performance terrain visualization," in *Proc. EG2003*, Sept. 2003, pp. 505–514.

[4] P. Cignoni, F. Ganovelli, E. Gobbetti, F. Marton, F. Ponchio, and R. Scopigno, "Adaptive TetraPuzzles – efficient out-of-core construction and visualization of gigantic polygonal models," *ACM Transactions on Graphics*, vol. 23, no. 3, August 2004.

[5] M. Duchaineau, M. Wolinsky, D. E. Sigeti, M. C. Miller, C. Aldrich, and M. B. Mineev-Weinstein, "ROAMing terrain: Real-time optimally adapting meshes," in *IEEE Visualization '97*, Nov. 1997, pp. 81–88.

[6] "Roam algorithm version 2.0—work in progress." [Online]. Available: http://www.cognigraph.com/ROAM_homepage/ROAM2/

[7] M. de Berg and K. T. G. Dobrindt, "On levels of detail in terrains," in *Proceedings of ACM Symposium on Computational Geometry*, June 1995, pp. C26–C27.

[8] P. J. Brown, "Selective mesh refinement for interactive terrain rendering," Cambridge University, Techinical Report, Computer Laboratory 417, Feb. 1997.

[9] D. Schmalstieg, "Smooth levels of detail," in *Proceedings of 1997 Virtual Reality Annual International Symposium*, Mar. 1997, pp. 12–19.

[10] A. A. Pomeranz, "ROAM using surface triangle clusters (RUSTiC)," Master's thesis, Center for Image Processing and Integrated Computing, University of California, Davis, 2000.

[11] W. S. Evans, D. G. Kirkpatrick, and G. Townsend, "Right-triangulated irregular networks," *Algorithmica*, vol. 30, no. 2, pp. 264–286, 2001.

[12] H. Hoppe, "Progressive meshes," in *Proceedings of SIGGRAPH 96*, ser. Computer Graphics Proceedings, Annual Conference Series, Aug. 1996, pp. 99–108.

[13] J. El-Sana and A. Varshney, "Generalized view-dependent simplification," *Computer Graphics Forum*, vol. 18, no. 3, pp. 83–94, Sept. 1999.

[14] J. Kim and S. Lee, "Transitive mesh space of a progressive mesh," *IEEE Transactions on Visualization and Computer Graphics*, vol. 9, no. 4, pp. 463–480, Oct.–Dec. 2003.

[15] H. Hoppe, "View-dependent refinement of progressive meshes," in *Proceedings of SIGGRAPH 97*, ser. Computer Graphics Proceedings, Annual Conference Series, Aug. 1997, pp. 189–198.

[16] ——, "Smooth view-dependent level-of-detail control and its application to terrain rendering," in *IEEE Visualization '98*, Oct. 1998, pp. 35–42.

[17] S.-E. Yoon, B. Salomon, R. Gayle, and D. Manocha, "Quick-vdr: Interactive view-dependent rendering of massive models," in *IEEE Vis 2004*. IEEE Computer Society, 2004, pp. 131–138.

[18] C. Erikson, D. Manocha, and W. V. B. III, "HLODs for faster display of large static and dynamic environments," in *2001 ACM Symposium on Interactive 3D Graphics*, Mar. 2001, pp. 111–120.

[19] Y. Zhu, "Uniform remeshing with an adaptive domain: A new scheme for view-dependent level-of-detail rendering of meshes," *IEEE Transactions on Visualization and Computer Graphics*, vol. 11, no. 3, pp. 301–316, Mar.–June 2005.

[20] S. Röttger, W. Heidrich, P. Slusallek, and H.-P. Seidel, "Real-time generation of continuous levels of detail for height fields," in *Proceedings of the 6th International Conference in Central Europe on Computer Graphics and Visualization*, Feb. 1998, pp. 315–322.

[21] R. Pajarola, "Large scale terrain visualization using the restricted quadtree triangulation," in *VIS '98: Proceedings of the conference on Visualization '98*. Los Alamitos, CA, USA: IEEE Computer Society Press, 1998, pp. 19–26.

[22] J. El-Sana and E. Bachmat, "Optimized view-dependent rendering for large polygonal datasets," in *IEEE Visualization 2002*, Oct.–Nov. 2002.

[23] Y. Zhu, "Real-time continuous level-of-detail terrain rendering with nested splitting space (Intel ISEF 2001 project, ID: CS012)," 2001. [Online]. Available: http://www.people.fas.harvard.edu/~yzhu/isef2k1-paper.pdf

[24] A. J. Stewart, "Hierarchical visibility in terrains," in *Eurographics Rendering Workshop 1997*, J. Dorsey and P. Slusallek, Eds. New York City, NY: Springer Wien, 1997, pp. 217–228.

[25] F. Corporation, "3dmark2001 second edition." [Online]. Available: http://www.futuremark.com/products/

[26] X. Bao, R. Pajarola, and M. Shafae, "SMART: An efficient technique for massive terrain visualization from out-of-core," in *Proceedings Vision, Modeling and Visualization (VMV)*, 2004, pp. 413–420.

[27] OpenGL Architecture Review Board, *OpenGL Reference Manual*, 2nd ed. Addison-Wesley, 1996.